**TEMA 6**

**POO Avanzada**

Contenido

[1. Herencia 2](#_Toc126096855)

[2. Clases derivadas 2](#_Toc126096856)

[2.1. Clases derivadas, Declaración 2](#_Toc126096857)

[2.2. Clases derivadas, Accesibilidad 2](#_Toc126096858)

[3. Constructores 3](#_Toc126096859)

[4. Sobreescritura de métodos 3](#_Toc126096860)

[4.1. Sobrecarga de métodos 3](#_Toc126096861)

[5. Conversión entre objetos derivados y objetos base 3](#_Toc126096862)

[6. Ligadura dinámica 4](#_Toc126096863)

[7. Métodos abstractos 4](#_Toc126096864)

# Herencia

La **Herencia** es una característica de la POO, que permite crear una clase, partiendo de otra ya creada, la nueva clase, hereda el estado y comportamiento de la clase base, con ello se consigue:

* Reutilización del código
* Definir la jerarquía de las clases
* Eliminar código repetitivo

La **herencia** permite producir software **adaptable y reutilizable,** Hay dos tipos de herencia

* **Herencia simple** solo puede heredar de una clase base (Java solo permite la simple)
* **Herencia múltiple** se puede heredar de más de una clase

# Clases derivadas

Las Clases derivadas, es una clase, que esta derivada, a otra, como por ejemplo programador y empleado, el **programador** es un tipo de **empleado** Diagrama

Descripción generada automáticamente

En este caso, la clase programador, es una extensión de Empleado

# Clases derivadas, Declaración

La clase derivada **extends**, hace que todos los miembros **no private** de la clase base, sean heredados en la clase derivada

# Clases derivadas, Accesibilidad

Una clase derivada no puede acceder a los miembros (atributos) **privados**, de su clase base, pero la clase base puede declarar atributos protegidos (**protected**) para ocultar detalles de la implementación

La clase derivada, no puede acceder a los atributos privados, solamente a **public** y **protected**

Diagrama

Descripción generada automáticamente

# Constructores

Los constructores de las clases base, son invocados, **antes** que los de las clases derivadas

La primera línea del constructor de la clase derivada, debe ser la llamada a un constructor de la clase base (**La llamada se realiza con el método super()**) si no se indica el método, se esta llamando al constructor por defecto (SIN ARGUMENTOS)

# Sobreescritura de métodos

El Overriding es una clase derivada, en la que es posible redefinir métodos heredados (Sobrescribirlos), tiene como finalidad añadir funcionalidad a la heredada por defecto

Para que se sobrescriba, debe cumplir:

* Tener el mismo nombre
* El tipo de retorno debe ser el mismo
* Tener el mismo número y tipo parámetros

# Sobrecarga de métodos

El Overloading es la implementación del mismo método varias veces, es un método que consiste el definir un método con el mismo nombre, pero distinto número o tipos de argumentos. El **tipo de retorno** no es tenido en cuentas en la sobrecarga, los métodos sobrecargados pueden cambiar el tipo de retorno , también se puede realizar en **clases derivadas**.

# Conversión entre objetos derivados y objetos base

Para saber de que tipo es el objeto, podemos utilizar **Instanceof**, son esenciales para implementar **métodos polimórficos**

Texto

Descripción generada automáticamente

# Ligadura dinámica

Una ligadura es una conexión entre la llamada a un método y el código que se ejecuta, puede ser de dos tipos:

* **Estática**: Si se realiza en tiempo de **compilación**
* **Dinámica:** Si se realiza en tiempo de **ejecución**

En la **ligadura estática**, el compilador y el enlazador definen directamente la posición del código

En la **ligadura dinámica**, el código a ejecutar no puede determinar hasta el momento de ejecución

La ligadura por defecto en Java, es la **estática**, la dinámica es cuando

* se redefinen los métodos
* se definen métodos abstractos

# Métodos abstractos

Un método abstracto es el declarado como tal en la clase base, con la palabra reservada **abstract** y será definido en una clase derivada, estos métodos están enfocados en el polimorfismo, si una clase derivada NO redefine algún método abstracto de su clase base, entonces se convierte en clase abstracta ya que hereda el método como abstracto.

La utilidad de una clase abstracta es utilizarla para fijar el comportamiento mínimo que debe tener una clase. En Java, abstract es sinónimo de genérico. Las clases abstractas son clases que han sido pensadas para ser genéricas. Crearemos entonces clases abstractas para representar conceptos generales, características comunes de un tipo de objetos. Esto quiere decir que no va a haber objetos de esas clases puesto que no tiene sentido

Las propiedades de una clase abstracta son:

* Se declara con la palabra reservada abstract.
* Una clase con al menos un método abstracto es una clase abstracta.
* Una clase derivada que no redefine un método abstracto heredado es también clase abstracta.
* Las clases abstractas pueden tener variables de instancia y métodos no abstractos.
* No se pueden crear objetos de clases abstractas, aunque si referencias

# Interfaces